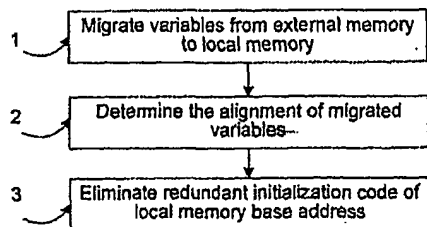
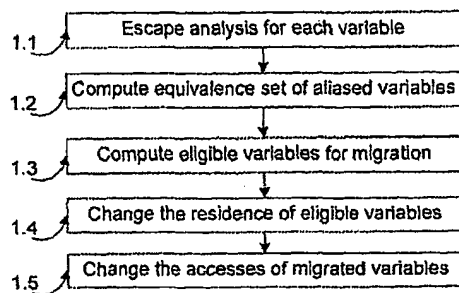
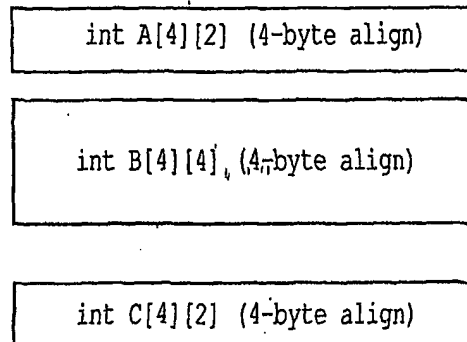


1/7

$$(BASE + OFFSET) == (BASE | OFFSET)$$

**Fig. 1****Fig. 2****Fig. 3**

2/7



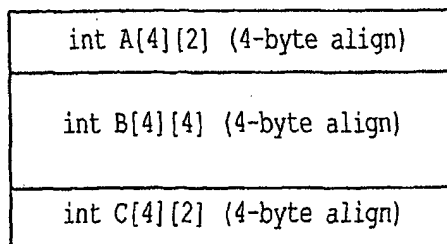
Original Data

```

1 Access Address A[i][0]
2 Access Address A[i][1]
3 Access Address B[i][0]
4 Access Address B[i][1]
5 Access Address B[i][2]
6 Access Address B[i][3]
7 Access Address A[i][0]
8 Access Address A[i][1]

```

Pseudo code sequence of accessing A, B, C

**FIG. 4A****FIG. 4B**

Data in local memory

```

Set the base address to A[i][0]
Access Address A[i][0] (A[i][0]+0)
Set the base address to A[i][1]
Access Address A[i][1] (A[i][1]+0)
Set the base address to B[i][0]
Access Address B[i][0] (B[i][0]+0)
Set the base address to B[i][1]
Access Address B[i][1] (B[i][1]+0)
Set the base address to B[i][2]
Access Address B[i][2] (B[i][2]+0)
Set the base address to B[i][3]
Access Address B[i][3] (B[i][3]+0)
Set the base address to C[i][0]
Access Address C[i][0] (C[i][0]+0)
Set the base address to C[i][1]
Access Address C[i][1] (C[i][1]+0)

```

Pseudo code sequence of  
accessing A, B, C with  
initialization code of local  
memory based address inserted

**FIG. 5A****FIG. 5B**

3/7

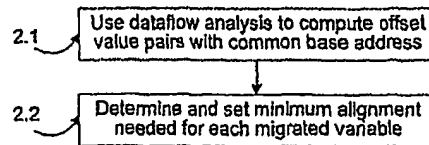


Fig. 6

```

Compute the GEN, KILL, IN, and OUT for each flow node, fill in the hash table (base address, set of offset value pair)
for (each base address in the hash table)
  VAR is the variable accessed by the base address
  for (each offset value pair for this base address)
    Int CURR_VAR_ALIGN = current alignment of VAR
    Int CURR_BASE_ALIGN = current alignment of the base address
    if (CURR_BASE_ALIGN does not satisfy the condition in Figure 1 for one of the offset value in the pair)
      Int NEEDED_BASE_ALIGN = the minimum base address alignment needed to satisfy the condition in Figure 1 for
      all offset values in the pair
      Int new_align = CURR_VAR_ALIGN * NEEDED_BASE_ALIGN / CURR_BASE_ALIGN
      if (new_align <= MAX_ALIGN(VAR))
        set the alignment of VAR to new_align
        if (the alignment change does not make the base address satisfy the condition in Figure 1 for all offset values in
        the pair)
          restore VAR's alignment to CURR_VAR_ALIGN
        end if
      end if
    end if
  end if
end if
  
```

Fig. 7

4/7

|                             |
|-----------------------------|
| int A[4][2] (8-byte align)  |
| int B[4][4] (16-byte align) |
| int C[4][2] (8-byte align)  |

Data with adjusted alignment

**FIG. 8A**

```

Set the base address to A[i][0]
Access Address A[i][0] (A[i][0]+0)
Set the base address to A[i][0]
Access Address A[i][1] (A[i][0]+4)
Set the base address to B[i][0]
Access Address B[i][0] (B[i][0]+0)
Set the base address to B[i][0]
Access Address B[i][1] (B[i][0]+4)
Set the base address to B[i][0]
Access Address B[i][2] (B[i][0]+8)
Set the base address to B[i][0]
Access Address B[i][3] (B[i][0]+12)
Set the base address to C[i][0]
Access Address C[i][0] (C[i][0]+0)
Set the base address to C[i][0]
Access Address C[i][1] (C[i][0]+4)

```

Pseudo code sequence of accessing A, B, C  
after insert code to initialize the  
local memory base address

**FIG. 8B**

|                             |
|-----------------------------|
| int A[4][2] (8-byte align)  |
| int B[4][4] (16-byte align) |
| int C[4][2] (8-byte align)  |

Data with adjusted alignment

**FIG. 9A**

```

Set the base address to A[i][0]
Access Address A[i][0] (A[i][0]+0)
Access Address A[i][1] (A[i][0]+4)
Set the base address to B[i][0]
Access Address B[i][0] (B[i][0]+0)
Access Address B[i][1] (B[i][0]+4)
Access Address B[i][2] (B[i][0]+8)
Access Address B[i][3] (B[i][0]+12)
Set the base address to C[i][0]
Access Address C[i][0] (C[i][0]+0)
Access Address C[i][1] (C[i][0]+4)

```

Pseudo code sequence of accessing A, B, C  
after insert code to initialize the  
local memory base address

**FIG. 9B**

5/7

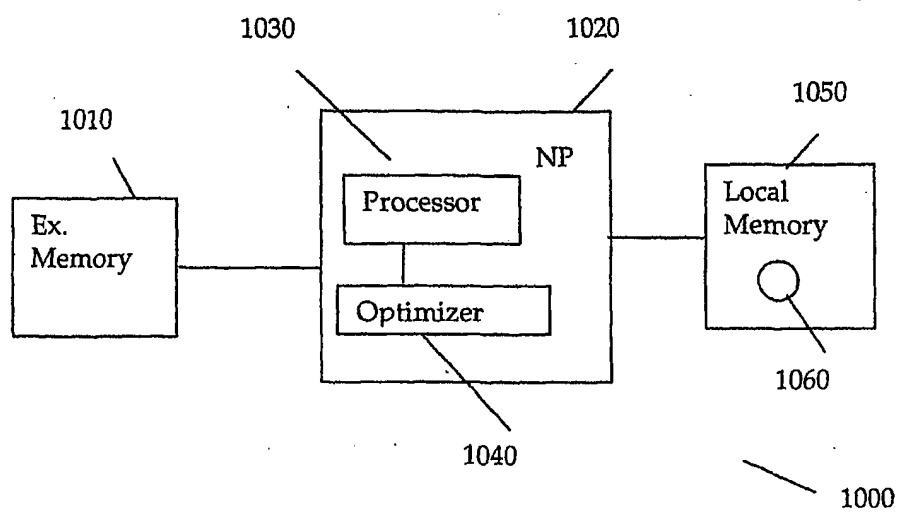


Fig. 10

6/7

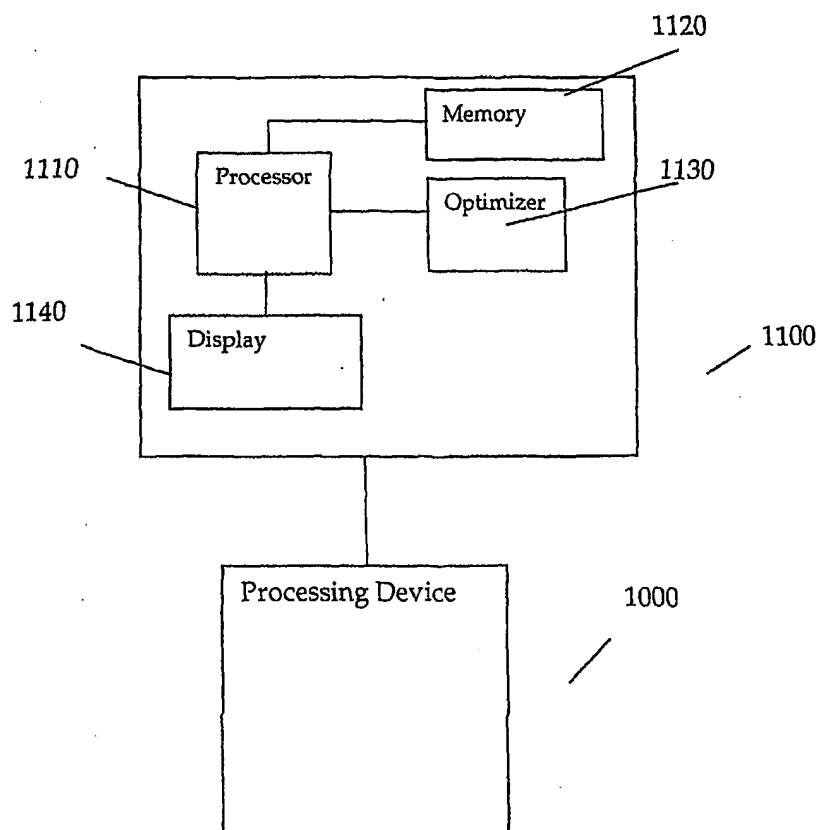


Fig. 11

7/7

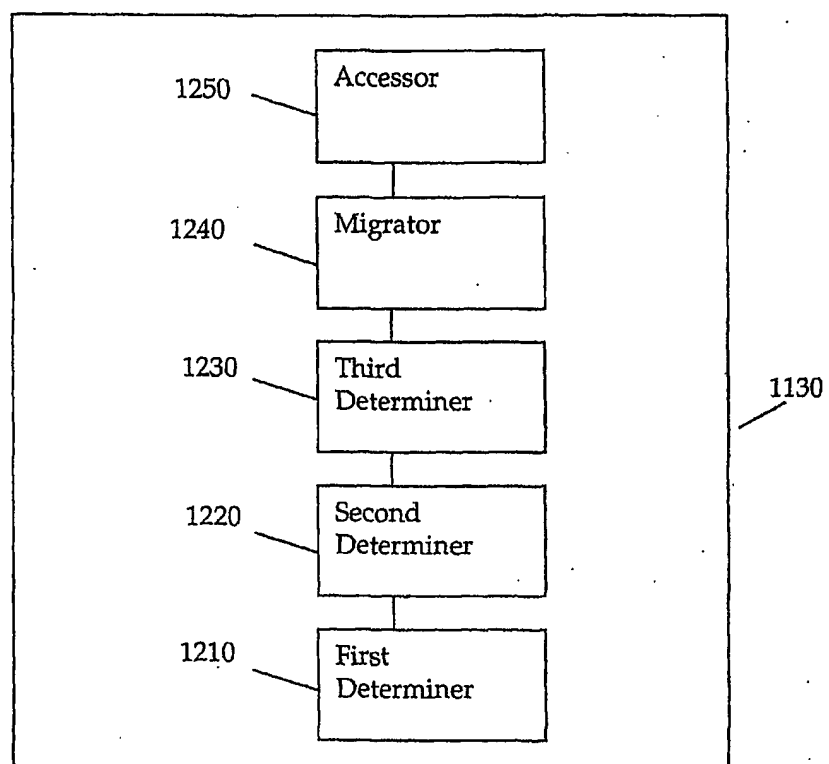


Fig. 12